Problem 1

$$\frac{\partial E}{\partial a_k} = -\sum_{n=1}^{N}\sum_{j=1}^{K} t_{nj}\frac{1}{y_j(\mathbf{x_n},\mathbf{w})}\frac{\partial y_j(\mathbf{x_n},\mathbf{w})}{\partial a_k} \text{ where } y_j = \frac{e^{a_j}}{\sum_{i=1}^{K} e^{a_i}}$$

$$(j=k)\frac{\partial y_j(\mathbf{x_n},\mathbf{w})}{\partial a_k} = \frac{e^{a_k}\sum_{i=1}^{K} e^{a_i} - e^{a_k}e^{a_k}}{\left[\sum_{i=1}^{K} e^{a_i}\right]^2} = y_k(1-y_k)$$

$$(j\neq k)\frac{\partial y_j(\mathbf{x_n},\mathbf{w})}{\partial a_k} = -\frac{e^{a_k}e^{a_j}}{\left[\sum_{i=1}^{K} e^{a_i}\right]^2} = -y_k y_j$$

$$\text{let } \delta_{jk} = \begin{cases} 1 & \text{if } j=k \\ 0 & \text{otherwise} \end{cases}$$

$$\Rightarrow \frac{\partial E}{\partial a_k} = -\sum_{n=1}^{N}\sum_{j=1}^{K} t_{nk}\frac{1}{y_j}y_j(\delta_{jk}-y_k)$$

$$= \sum_{n=1}^{N}\sum_{j=1}^{K} t_{nj}(y_k-\delta_{jk})$$

$$\sum_{j=1}^{K} t_{nj} = 1 \Rightarrow \sum_{j=1}^{K} t_{nj}y_k = y_k \quad t_n \text{ is a one-of-K coding scheme.}$$

$$\sum_{j=1}^{K} t_{nj}\delta_{jk} = t_{nk}\delta_{kk} = t_{nk} \quad \delta_{jk} = 1 \text{ if } j=k \text{ and } \delta_{jk} = 0 \text{ otherwise.}$$

$$\text{Thus, } \frac{\partial E}{\partial a_k} = \sum_{n=1}^{N} y_k - t_{nk}$$

Problem 2

Designed a neural net that contained 10 units in its hidden layer, used a batch size of 10, a learning rate of 0.001, a tolerance of 0.1, and 1000 as the maximum number of iterations. Obtained a 99.5% accuracy on training data and 110% accuracy on the test data.

Neural Nets Implementation & XOR Problem

In [33]:
```
import numpy as np
import pickle
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [14]:
```
# Load xor toy dataset
with open('xordata.pkl', 'rb') as f:
    data = pickle.load(f)

X_train = data['X_train'] # 800 training data points with 2 features
y_train = data['y_train'] # training binary labels {0,1}

X_test = data['X_test']
y_test = data['y_test']
```

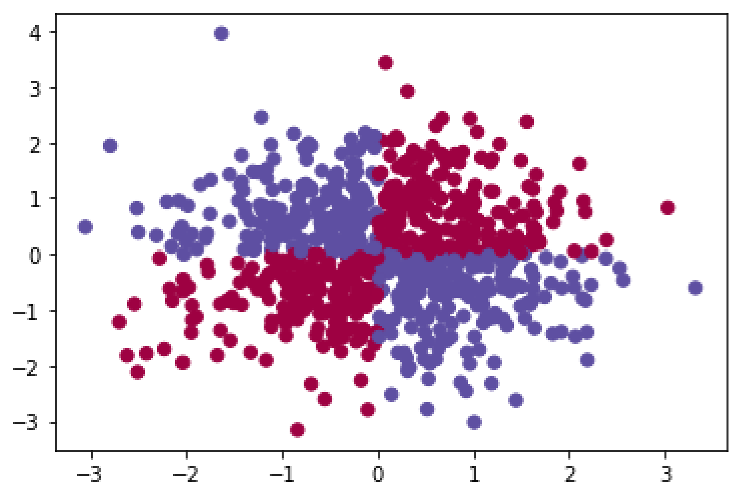
In [15]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(800, 2) (800,)
(200, 2) (200,)
```

In [16]:
```
X_val,X_test,y_val,y_test=train_test_split(X_test,y_test,test_size=0.5)
```

In [7]:
```
plt.scatter(X_train[:,0], X_train[:,1], s=40, c=y_train, cmap=plt.cm.Spectral)
```

Out[7]: <matplotlib.collections.PathCollection at 0x7fabf133ee80>



Instructions:

Goal: implement from scratch brackprop to train a simple neural network and test it on a simple dataset.

- Implement brackprop to train a two-layer perceptron: an input layer, a hidden layer, and an output layer.
- The core of the code should include: a forward pass; a backward pass; weight updates.
- For input and output layers specify the number of nodes appropriate for the above problem.
- Randomly initialize the weights and biases of the network.
- For the hidden layer use ReLU as an activation function and for the output layer use logistic sigmoid.
- Use cross-entropy loss as the network's loss function and mini-batch SGD as the optimizer.
- Feel free to tune the network as you see fit (including number of units in the hidden layer, learning rate, batch size, number of epochs, etc).
- (Optional) You can use sklearn.inspection.DecisionBoundaryDisplay to visualize your decision boundary.
- Report the accuracy of the network on the train and test set. Remember to create and use a validation set for the training phase!

In [34]:

```
def relu(x):
    return np.maximum(0,x)

def relu_derivative(x):
    return np.where(x>0,1,0)

def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x)*(1-sigmoid(x))

class TwoLayerPerceptron:
    def __init__(self,input_dim,hidden_dim,learning_rate=0.001,batch_size=10,epc
        self.learning_rate=learning_rate
        self.epochs=epochs
        self.batch_size=batch_size
        self.tol=tol
        self.W1,self.b1,self.W2,self.b2=self.initialize_parameters(input_dim,hic

    def initialize_parameters(self,input_dim,hidden_dim):
        W1=np.random.randn(input_dim,hidden_dim)
        b1=np.random.randn(hidden_dim)
        W2=np.random.randn(hidden_dim,1)
        b2=np.random.randn(1)
        return W1,b1,W2,b2

    def forward_pass(self,X):
        Z1=np.dot(X,self.W1)+self.b1
        A1=np.apply_along_axis(relu,0,Z1)
        Z2=np.dot(A1,self.W2)+self.b2
        A2=sigmoid(Z2)
        return Z1,A1,Z2,A2

    def backward_prop(self,X,y,Z1,A1,Z2,A2):
        m=X.shape[0]
        dA2=A2-np.reshape(y,(-1,1))
        dZ2=dA2*sigmoid_derivative(Z2)
        dW2=(1/m)*np.dot(A1.T,dZ2)
```

```
        db2=(1/m)*np.sum(dZ2)

        dA1=np.dot(dZ2,self.W2.T)
        dZ1=dA1*relu_derivative(Z1)
        dW1=(1/m)*np.dot(X.T,dZ1)
        db1=(1/m)*np.sum(dZ1,axis=0)
        return dW1,db1,dW2,db2

    def update_weights(self,dW1,db1,dW2,db2):
        self.W1-=self.learning_rate*dW1
        self.b1-=self.learning_rate*db1
        self.W2-=self.learning_rate*dW2
        self.b2-=self.learning_rate*db2

    def cross_entropy_loss(self,X,y):
        _,_,_,y_pred=self.forward_pass(X)
        entropy=np.where(y==1,np.log(y_pred.flatten()),np.log(1-y_pred.flatten()
        return -np.sum(entropy)

    def predict(self, X):
        output=self.forward_pass(X)[3].flatten()
        predictions = (output > 0.5).astype(int)
        return predictions

    def accuracy(self, X,y_true):
        y_pred=self.predict(X)
        return np.mean(y_true == y_pred)

    def train(self,X,y):
        for epoch in np.arange(self.epochs):
            if self.cross_entropy_loss(X_val,y_val)<self.tol:
                break

            permutation = np.random.permutation(X_train.shape[0])
            X_train_shuffled = X_train[permutation]
            y_train_shuffled = y_train[permutation]

            for i in range(0, X_train.shape[0], self.batch_size):
                X_batch = X_train_shuffled[i:i + self.batch_size]
                y_batch = y_train_shuffled[i:i + self.batch_size]

                Z1, A1, Z2, A2 = self.forward_pass(X_batch)
                dW1, db1, dW2, db2 = self.backward_prop(X_batch, y_batch, Z1,A1,
                self.update_weights(dW1, db1, dW2, db2)
```

In [35]:

```
model=TwoLayerPerceptron(2,10,1)
```

In [36]:

```
model.train(X_train,y_train)
```

```
<ipython-input-34-283db22abf63>:56: RuntimeWarning: divide by zero encountered i
n log
  entropy=np.where(y==1,np.log(y_pred.flatten()),np.log(1-y_pred.flatten()))
```

In [37]:

```
model.accuracy(X_test,y_test)
```

Out[37]: 1.0

In [38]:
```
model.cross_entropy_loss(X_test,y_test)
```

```
<ipython-input-34-283db22abf63>:56: RuntimeWarning: divide by zero encountered i
n log
  entropy=np.where(y==1,np.log(y_pred.flatten()),np.log(1-y_pred.flatten()))
```

Out[38]: 0.02884181588548293

In [39]:
```
model.accuracy(X_train,y_train)
```

Out[39]: 0.995

In [ ]: